Introduction to C++ Programming

Cecina Babich Morrow

2024-01-29

Introduction to *C++* Programming

C++ is a general-purpose programming language that offers power, efficiency, and speed. Because it is a compiled language, it is much faster than interpreted languages like Python and R. The downside is that it is a complex language with a steeper learning curve. This trade-off means that C++ is most likely to be the "right" choice for a project when you are more focused on minimizing computation time than on minimizing development time.

Basic C++ Syntax

Since C++ is a compiled language, once you have written a C++ program, you need to compile it to create an executable that can run on your computer. You can run the following command in your terminal to use the compiler `g++` to create an executable program from a given C++ file:

```
g++ -o your_file_name.cpp -o your_file_name
```

To run the program, run `./your_file_name` in your terminal.

Below is a basic Hello, World! program in C++:

```
#include <iostream>
#include <string>

int main()
{
    /* we can write
        multi-line comments
          like this */
    std::string a = "Hello";
    std::string b = "from";
    std::string c = "C++!";

    // Single line comments as so
    std::cout << a << " " << b << " " << c << std::endl;

    return 0;
}
```

Note the following C++ syntax elements: Each line must end with `;`. All C++ programs must include a `main` function, which is executed when the program is run. `main` returns an integer value representing the status of the program (`0` if the program ran correctly). Header files are included at the beginning of the program, such as `#include <iostream>` above. These include a given set of functions, classes, and objects.

Types

Variable types are crucial in C++. Every variable must have its type specified the first time it is declared, such as `std::string` above. Variable types cannot be changed after they are declared (unlike in Python and R). It is however possible to perform type conversions so that the data held in a variable of one type can be held in a new variable of a different type, such as converting an `int` to a `double`.

For C++11 onwards, the `auto` keyword can be used to automatically determine the type of a variable based on the value to which it is first assigned. For example, `auto x = 5;` will declare `x` as an `int` because `5` is an `int`.

Conditionals

Conditional statements are written in C++ using the following syntax:

```cpp
#include <iostream>

int main()
{
    int i = -2;

    if (i < 10)
    {
        std::cout << "i is less than 10" << std::endl;
    }
    else if (i >= 100)
    {
        std::cout << "i is more than or equal to 100" << std::endl;
    }
    else
    {
        std::cout << "i is somewhere between 10 and 100" << std::endl;
    }

    return 0;
}
```

Functions

The syntax for writing functions in C++ is as follows in this set of example functions:

```cpp
#include <iostream>

int sum(int a, int b)
{
    int c = a + b;
    return c;
}

void print(std::string s)
{
    std::cout << s << std::endl;
}
```

```cpp
double square(double x)
{
    return x * x;
}

std::string join(std::string a, std::string b)
{
    return a + " " + b;
}

void print_hello()
{
    std::cout << "Hello World" << std::endl;
}

int main()
{
    std::cout << "5 + 3 equals " << sum(5, 3) << std::endl;

    print("Hello from a function!");

    std::cout << "The square of 3.5 is " << square(3.5) << std::endl;

    std::cout << join("Hello", "World") << std::endl;

    print_hello();

    // you can pass the return value from one function as the argument
    // of another, e.g.
    print(join("Hello", join("from", "C++")));

    return 0;
}
```

Note that you must specify the type of the return value of the function. (If the function does not return a value, the return type is `void`, as in `print_hello` above.)

A function can only been used in a file after it has been declared. In the code above, all of the functions are defined before they are called. However, it is possible to declare a function (without defining it), call it, and then define it. The declaration of a function is written as the function name followed by the types of the arguments in parentheses, followed by a semicolon. For example, the function prototype for the `sum` function above is `int sum(int a, int b);`. Functions can be declared multiple times, but only defined once.

Variable scope

The scope of a variable is the part of the program where that variable can be accessed. The scope of a variable is determined by where it is declared. Variables declared inside a set of curly brackets `{}` can only be accessed within those brackets. A variable can only be declared once within a given scope, but it is possible to declare a variable with the same name in different scopes.

Vectors

C++ vectors can contain a list of values accessible by index. In C++, vectors can only hold values of a single type. The following example creates a vector `v` that can hold integers and appends the values `4` and `2` onto the end of the vector:

```cpp
#include <iostream>
#include <vector>

int main()
{
    // create a vector that can hold integers
    std::vector<int> v;

    // append values onto the end of the vector
    v.push_back(4);
    v.push_back(2);

    // .size() returns the number of elements in the vector
    std::cout << "The size of the vector v is " << v.size() << std::endl;

    // square brackets access a given element of the vector
    // vectors are 0-indexed
    std::cout << "The first element of the vector v is " << v[0] << std::endl;

    return 0;
}
```

Note that you need to include the `<vector>` header file to use vectors in C++.

As an alternative to using `push_back` to add every element to the vector individually, you can create a vector with a given set of values using `{}`:

```cpp
#include <iostream>
#include <vector>

int main()
{
    // create a vector that can hold integers
    std::vector<int> v = { 1, 4, 9, 16, 25, 36, 49, 64, 81, 100 };

    std::cout << "The size of the vector v is " << v.size() << std::endl;

    std::cout << "The first element of the vector v is " << v[0] << std::endl;

    return 0;
}
```

Loops

Loops are written in C++ in a manner following this example:

```cpp
#include <iostream>

int main()
{
    for (float i=1; i<=10; i=i+0.5)
    {
        std::cout << i << " times 5 equals " << (i*5) << std::endl;
    }

    return 0;
}
```

Following our discussion of variable scope, it is interesting to note that variables declared in a for loop's initializer are only accessible within the loop. For instance `i` above is only accessible within that loop.

We can also loop over elements of a vector directly:

```cpp
#include <iostream>
#include <vector>

void print_int_vec(std::vector<int> v) {

    for(auto e : v) {
        std::cout << e << " ";
    }

    std::cout << std::endl;
}

int main() {

    std::vector<int> v = {4, 2, 9, 12, 13};

    print_int_vec(v);

    return 0;
}
```

Maps

C++ maps, similar to Python dictionaries, store key-value pairs. In C++, all the keys must have the same type and all the values must have the same type (but keys and values do not need to be the same type). The following example creates a map `database` storing people's height and weight and uses a function called `bmi` to calculate the BMI of each person, then adds that data to the map:

```cpp
#include <iostream>
#include <string>
#include <map>
#include <vector>

float bmi(float weight, float height)
{
    float bmi = weight / (height * height);
```

```
    return bmi;
}

int main()
{
    // declare the map that uses a person's name as a key, and looks
    // up a map that stores the person's weight and height
    std::map< std::string, std::map<std::string, float> > database;

    // let's first put the data in three vectors
    std::vector<std::string> names = { "James", "Jane", "Janet", "John" };
    std::vector<float> heights = { 1.7, 1.8, 1.5, 1.4 };
    std::vector<float> weights = { 75.4, 76.5, 56.8, 52.0 };

    // now put all of the data into the database
    for (int i=0; i<names.size(); ++i)
    {
        std::map<std::string,float> data;

        data["height"] = heights[i];
        data["weight"] = weights[i];

        database[names[i]] = data;
    }

    // Calculate bmi for every person and save it in the database
    for (int j = 0; j < names.size(); j++)
    {
        float bmi_j = bmi(weights[j], heights[j]);
        database[names[j]]["bmi"] = bmi_j;
    }


    // now print out the entire database
    for ( auto item : database )
    {
        // print out the name
        std::cout << item.first << " : ";

        // now print out all of the data about the person
        for ( auto data : item.second )
        {
            std::cout << data.first << "=" << data.second << " ";
        }

        std::cout << std::endl;
    }

    return 0;
}
```